Example for collinear factorization in p+p collisions

In this example, we study single inclusive π^0 production in $p+p$ collisions: $p+p \to \pi^0 + X$. Within the collinear factorization formalism, the differential cross section can be written as follows

$$\frac{d\sigma}{dy\, d^2p_T} = \frac{\alpha_s^2}{S} \sum_{a,b,b} \int \frac{dx_a}{x_a} f_{a/p}(x_a,\mu) \int \frac{dx_b}{x_b} f_{b/p}(x_b,\mu) \int \frac{dz_c}{z_c^2} D_{\pi^0/c}(z_c,\mu)\, H_{ab\to c}(\hat{s},\hat{t},\hat{u})\, \delta(\hat{s}+\hat{t}+\hat{u})$$

load proper packages

1. We need to use LHAPDF to call collinear parton distribution functions (PDFs) $f_{a/p}(x,\mu)$ and collinear fragmentation functions (FFs) $D_{\pi^0/c}(z,\mu)$. Follow their website information to install LHAPDF.
2. At the same time, we also have to use VEGAS Monte Carlo integration. One can install through "pip install vegas" at command line

LHAPDF installation on Google Colab

```
LHAPDF_VERSION = '6.5.4'
SITE_PACKGE_DIR = __import__('site').getsitepackages()[0]
PWD = __import__('os').getcwd()
PYTHON_VERSION = '%s.%s' % __import__('sys').version_info[0:2]
!wget https://lhapdf.hepforge.org/downloads/?f=LHAPDF-{LHAPDF_VERSION}.tar.gz -O LHAPDF-{LHAPDF_VERSION}.tar.gz
!tar xf LHAPDF-{LHAPDF_VERSION}.tar.gz
!cd LHAPDF-{LHAPDF_VERSION} && ./configure
!make -C LHAPDF-{LHAPDF_VERSION} -j 2
!make -C LHAPDF-{LHAPDF_VERSION} install
!cd {SITE_PACKGE_DIR} && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/wrappers/python/NONE/local/lib/python{PYTHON_VERSION}/dist-packag
!cd /usr/lib && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/src/.libs/libLHAPDF.so
!lhapdf update
```

```
make[3]: Leaving directory '/content/LHAPDF-6.5.4'
make[2]: Leaving directory '/content/LHAPDF-6.5.4'
make[1]: Leaving directory '/content/LHAPDF-6.5.4'
make: Leaving directory '/content/LHAPDF-6.5.4'
pdfsets.index:    46.5 KB[100.0%]
```

```
import lhapdf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
# LHAPDF_VERSION = '6.5.4'
# SITE_PACKGE_DIR = __import__('site').getsitepackages()[0]
# PWD = __import__('os').getcwd()
# PYTHON_VERSION = '%s.%s' % __import__('sys').version_info[0:2]
# !wget https://lhapdf.hepforge.org/downloads/?f=LHAPDF-{LHAPDF_VERSION}.tar.gz -O LHAPDF-{LHAPDF_VERSION}.tar.gz
# !tar xf LHAPDF-{LHAPDF_VERSION}.tar.gz
# !cd LHAPDF-{LHAPDF_VERSION} && ./configure
# !make -C LHAPDF-{LHAPDF_VERSION} -j 2
# !make -C LHAPDF-{LHAPDF_VERSION} install
# !cd {SITE_PACKGE_DIR} && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/wrappers/python/NONE/local/lib/python{PYTHON_VERSION}/dist-pac
# !cd /usr/lib && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/src/.libs/libLHAPDF.so
# !lhapdf update
```

```
LHAPDF_VERSION = '6.5.4'
SITE_PACKGE_DIR = __import__('site').getsitepackages()[0]
PWD = __import__('os').getcwd()
PYTHON_VERSION = '%s.%s' % __import__('sys').version_info[0:2]
#!wget https://lhapdf.hepforge.org/downloads/?f=LHAPDF-{LHAPDF_VERSION}.tar.gz -O LHAPDF-{LHAPDF_VERSION}.tar.gz
#!tar xf LHAPDF-{LHAPDF_VERSION}.tar.gz
#!cd LHAPDF-{LHAPDF_VERSION} && ./configure
#!make -C LHAPDF-{LHAPDF_VERSION} -j 2
#!make -C LHAPDF-{LHAPDF_VERSION} install
!cd {SITE_PACKGE_DIR} && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/wrappers/python/NONE/local/lib/python{PYTHON_VERSION}/dist-packa
#!cd /usr/lib && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/src/.libs/libLHAPDF.so
#!lhapdf update

! echo SITE_PACKGE_DIR
```

```
ln: failed to create symbolic link './lhapdf': File exists
SITE_PACKGE_DIR
```

```
!cd {SITE_PACKGE_DIR}
! ls
```

```
LHAPDF-6.5.4  LHAPDF-6.5.4.tar.gz  sample_data
```

```
! cd ../.config
! pwd
! ls -a
```

```
/bin/bash: line 1: cd: ../.config: No such file or directory
/content
.  ..  .config  LHAPDF-6.5.4  LHAPDF-6.5.4.tar.gz  sample_data
```

```
# LHAPDF_VERSION = '6.5.4'
# SITE_PACKGE_DIR = __import__('site').getsitepackages()[0]
# PWD = __import__('os').getcwd()
# PYTHON_VERSION = '%s.%s' % __import__('sys').version_info[0:2]
# !wget https://lhapdf.hepforge.org/downloads/?f=LHAPDF-{LHAPDF_VERSION}.tar.gz -O LHAPDF-{LHAPDF_VERSION}.tar.gz
# !tar xf LHAPDF-{LHAPDF_VERSION}.tar.gz
# !cd LHAPDF-{LHAPDF_VERSION} && ./configure
# !make -C LHAPDF-{LHAPDF_VERSION} -j 2
# !make -C LHAPDF-{LHAPDF_VERSION} install
# !cd {SITE_PACKGE_DIR} && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/wrappers/python/NONE/local/lib/python{PYTHON_VERSION}/dist-pac
# !cd /usr/lib && ln -s {PWD}/LHAPDF-{LHAPDF_VERSION}/src/.libs/libLHAPDF.so
# !lhapdf update
```

```
! pwd
```

```
/content
```

```
! lhapdf-config
```

```
lhapdf-config: configuration tool for the LHAPDF
               parton density function evolution library
               http://projects.hepforge.org/lhapdf/

Usage: lhapdf-config [options]
```

```
Options:
  --help | -h   : show this help message
  --prefix      : show the installation prefix (cf. autoconf)
  --incdir      : show the path to the LHAPDF C++ header directory
  --libdir      : show the path to the LHAPDF library directory
  --datadir     : show the path to the LHAPDF data directory

  --cxx         : get compiler (including -std=c++11 flag or equiv)
  --cflags      : get compiler flags (aka --cppflags|--cxxflags)
  --libs        : get linker flags (aka --ldflags)

  --version     : return LHAPDF release version number
```

Install specific PDF set (CTEQ6L1) and FF set (MAPFF for pion)

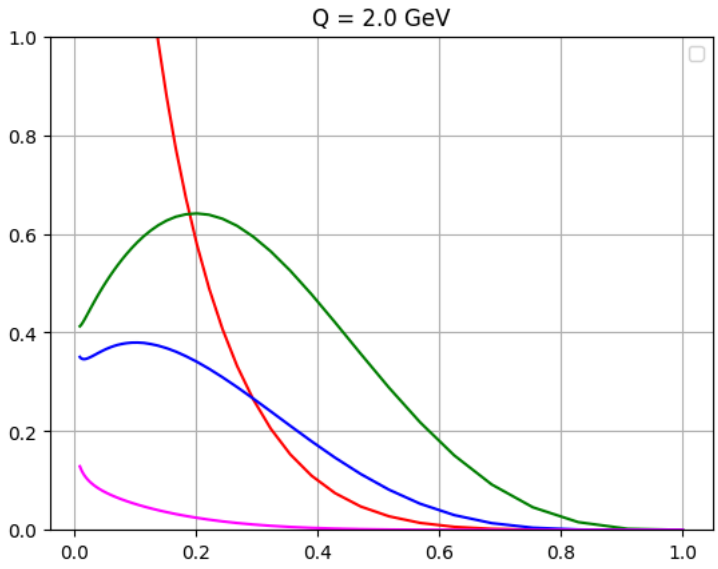
```
!lhapdf install cteq6l1
```

```
cteq6l1.tar.gz:   310.4 KB[100.0%]
/usr/local/bin/lhapdf:174: DeprecationWarning: Python 3.14 will, by default, filter extracted tar archives and reject files or mod
  tar_file.extractall(dest_dir)
```

```
Start coding or generate with AI.
```

```
# call the central number (0) of CTEQ6L1
p = lhapdf.mkPDF("cteq6l1/0")

# let us look at the flavors, they follow convention of particle data group
# d = 1, u = 2, s = 3, c = 4, b = 5, g = 21
print('flavor:', p.flavors())
# lhapdf gives by default - xfxQ2: x * f(x, Q^2)
#                       or - xfxQ:  x * f(x, Q)

q = 2.0
xx = np.geomspace(0.01, 1)

gg = [p.xfxQ(0, x, q) for x in xx]
dd = [p.xfxQ(1, x, q) for x in xx]
uu = [p.xfxQ(2, x, q) for x in xx]
ss = [p.xfxQ(3, x, q) for x in xx]

plt.plot(xx, gg, 'red', label='')
plt.plot(xx, uu, 'green', label='')
plt.plot(xx, dd, 'blue', label='')
plt.plot(xx, ss, 'magenta', label='')
plt.title(f'Q = {q} GeV')
plt.xlabel('')
plt.ylabel('')
plt.legend()
plt.grid()
plt.ylim(0, 1)
plt.show()
plt.close()
```

```
flavor: [-5, -4, -3, -2, -1, 1, 2, 3, 4, 5, 21]
/tmp/ipykernel_3102/3939310797.py:25: UserWarning: No artists with labels found to put in legend.  Note that artists whose label s
  plt.legend()
```



FFs: MAP pion fragmentation function at leading-order

In their convention, pisum = $\pi^+ + \pi^-$. From isospin symmetry, we have $\pi^0 = (\pi^+ + \pi^-)/2$. This is why we divide it by 2.

```
!lhapdf install MAPFF10NLOPIsum
```

```
MAPFF10NLOPIsum.tar.gz:    65.9 MB[100.0%]
/usr/local/bin/lhapdf:174: DeprecationWarning: Python 3.14 will, by default, filter extracted tar archives and reject files or mod
  tar_file.extractall(dest_dir)
```

```
pisum = lhapdf.mkPDF("MAPFF10NLOPIsum/0")
pisum.flavors()

x = 0.3
Q = 10.0
bbf= pisum.xfxQ(-5, x, Q)/2.0
cbf= pisum.xfxQ(-4, x, Q)/2.0
sbf= pisum.xfxQ(-3, x, Q)/2.0
ubf= pisum.xfxQ(-2, x, Q)/2.0
dbf= pisum.xfxQ(-1, x, Q)/2.0
df = pisum.xfxQ(1, x, Q)/2.0
uf = pisum.xfxQ(2, x, Q)/2.0
sf = pisum.xfxQ(3, x, Q)/2.0
cf = pisum.xfxQ(4, x, Q)/2.0
bf = pisum.xfxQ(5, x, Q)/2.0
gf = pisum.xfxQ(21, x, Q)/2.0
print('%0.2e, %0.2e, %0.2e, %0.2e, %0.2e' %(bbf,cbf,sbf,ubf,dbf))
print('%0.2e, %0.2e, %0.2e, %0.2e, %0.2e, %0.2e' %(df,uf,sf,cf,bf,gf))
```

```
1.13e-01, 2.55e-01, 2.73e-01, 2.39e-01, 2.06e-01
2.06e-01, 2.39e-01, 2.73e-01, 2.55e-01, 1.13e-01, 1.19e-01
```

For more convenient use below, I will define PDF and FF function

```
# construct light flavor PDFs and FFs
# note: for now, we only need PDFs and FFs of quarks
def PDF(x, Q):

    u = p.xfxQ(2, x, Q)/x
    d = p.xfxQ(1, x, Q)/x
    ub= p.xfxQ(-2, x, Q)/x
    db= p.xfxQ(-1, x, Q)/x
    s = p.xfxQ(3, x, Q)/x
    g = p.xfxQ(21, x, Q)/x

    return u,d,ub,db,s,g

def FF(hh, x, Q):
    fu = hh.xfxQ(2, x, Q)/x/2.0
    fub= hh.xfxQ(-2, x, Q)/x/2.0
    fd = hh.xfxQ(1, x, Q)/x/2.0
    fdb= hh.xfxQ(-1, x, Q)/x/2.0
    fs = hh.xfxQ(3, x, Q)/x/2.0
    fsb= hh.xfxQ(-3, x, Q)/x/2.0
    fg = hh.xfxQ(21, x, Q)/x/2.0

    return fu,fub,fd,fdb,fs,fsb,fg
```

```
# from google.colab import drive
# drive.mount('/content/drive')
```

I also need VEGAS Monte Carlo integration routine

```
!pip install vegas
```

```
Collecting vegas
  Downloading vegas-6.4.1-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (2.2 kB)
Requirement already satisfied: numpy>=1.24 in /usr/local/lib/python3.12/dist-packages (from vegas) (2.0.2)
Requirement already satisfied: scipy>=1.11 in /usr/local/lib/python3.12/dist-packages (from vegas) (1.16.3)
Collecting gvar>=13.1.5 (from vegas)
  Downloading gvar-13.1.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl.metadata (2.2 kB)
Downloading vegas-6.4.1-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (4.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.2/4.2 MB 27.3 MB/s eta 0:00:00
Downloading gvar-13.1.9-cp312-cp312-manylinux2014_x86_64.manylinux_2_17_x86_64.manylinux_2_28_x86_64.whl (7.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.8/7.8 MB 47.4 MB/s eta 0:00:00
Installing collected packages: gvar, vegas
Successfully installed gvar-13.1.9 vegas-6.4.1
```

Strong coupling constant

At leading order, strong coupling constant α_s runs through the following equation

$$\alpha_s(Q) = \frac{4\pi}{\beta_0 \ln\left(Q^2/\Lambda^2_{\rm QCD}(n_f)\right)},$$

where $\beta_0 = 11 - 2/3 n_f$ with n_f the active flavor of quarks. $\Lambda_{\rm QCD}$ in general also depends on n_f. Here the values is quoted from CTEQ6L1.

```
def alpha(Q):
    mb = 4.5
    if(Q <= mb):
        nf = 4
        lam_QCD = 0.215
    else:
        nf = 5
        lam_QCD = 0.165

    b0 = 11.0 - 2.0/3.0*nf
    tt = 2.0*np.log(Q/lam_QCD)
    return 4.0*np.pi/(b0*tt)
```

Partonic cross section

These are $H_{ab\to c}(\hat{s}, \hat{t}, \hat{u})$ given above.

```
def Upp(s, t, u):
    Nc = 3.0
#   qq' --> qq'
    WQ1=(Nc**2-1.0)/(2.0*Nc**2)*(s*s+u*u)/(t*t)
#   qq  ->qq
    WQ2=(Nc**2-1.0)/(2.0*Nc**2)*( (s*s+u*u)/(t*t)+(s*s+t*t)/(u*u) ) \
         -(Nc**2-1.0)/(Nc**3)*(s*s)/(t*u)
#   qqb ->q'qb'
    WQ3=(Nc**2-1.0)/(2.0*Nc**2)*(t*t+u*u)/(s*s)
#   qqb ->qqb
    WQ4=(Nc**2-1.0)/(2.0*Nc**2)*( (s*s+u*u)/(t*t)+(t*t+u*u)/(s*s) ) \
        -(Nc**2-1.0)/(Nc**3)*(u*u)/(s*t)
#   qqb ->gg
    WQ5=(Nc**2-1.0)**2/(2.0*Nc**3)*(u/t+t/u) \
    -(Nc**2-1.0)/Nc*(t*t+u*u)/(s*s)
#   gg  ->qqb
    WQ6=1.0/(2.0*Nc)*(t/u+u/t)-Nc/(Nc**2-1.0)*(t*t+u*u)/(s*s)
#   qg  ->qg
    WQ7=(Nc**2-1.0)/(2.0*Nc**2)*(-s/u-u/s)+(s*s+u*u)/(t*t)
#   gg  ->gg
    WQ8=4.0*Nc**2/(Nc**2-1.0)*(3.0-t*u/(s*s)-s*u/(t*t)-s*t/(u*u))

    return WQ1, WQ2, WQ3, WQ4, WQ5, WQ6, WQ7, WQ8
```

Below is the differential cross section

1. Since the theory formula has a delta-function $\delta\left(\hat{s} + \hat{t} + \hat{u}\right)$, one can use it to integrate out the variable x_b.
2. Then we will have integration over x_a and z_c.
3. If the experiment further measures the cross section for a specific range of rapidity y and transverse momentum p_T, we have to integrate over those ranges.
4. We also have to choose the factorization scale μ over there. In the program below, this μ is called `scale`. Usually, we choose `scale` to be the typical hard scale of the process. In our situation, this should be transverse momentum of the pion.
5. This is why we write the cross section as a function of x_a, z_c, y, p_T and `scale`.
6. There are so many partonic channels you have to sum over. This is the $\sum_{a,b,c}$ in the theory formula given above.

```
def sigma(xa, zc, y, roots, pT, scale):
    roots2 = roots**2
    AA=xa*roots2-pT/zc*roots*np.exp(y)
    BB=xa*pT/zc*roots*np.exp(-y)
    xb=BB/AA

    sh=xa*xb*roots2
    uh=-pT/zc*xb*roots*np.exp(y)
    th=-pT/zc*xa*roots*np.exp(-y)
```

```
    if(xa < 1. and xa > 0. and xb < 1. and xb > 0. and zc < 1. and zc > 0.):
        U1,D1,UB1,DB1,S1,GL1 = PDF(xa, scale)
        U2,D2,UB2,DB2,S2,GL2 = PDF(xb, scale)
        fu,fub,fd,fdb,fs,fsb,fg = FF(pisum, zc, scale)
        WQ1,WQ2,WQ3,WQ4,WQ5,WQ6,WQ7,WQ8 = Upp(sh,th,uh)
        WT1,WT2,WT3,WT4,WT5,WT6,WT7,WT8 = Upp(sh,uh,th)

        SIG1=WQ2*(U1*U2*fu+UB1*UB2*fub+D1*D2*fd+DB1*DB2*fdb+S1*S2*fs+S1*S2*fsb) \
            +WQ3*(U1*UB2*(fd+fs)+D1*DB2*(fu+fs)+S1*S2*(fu+fd) \
            +UB1*U2*(fdb+fsb)+DB1*D2*(fub+fsb)+S1*S2*(fub+fdb)) \
            +WT3*(UB1*U2*(fd+fs)+DB1*D2*(fu+fs)+S1*S2*(fu+fd) \
            +U1*UB2*(fdb+fsb)+D1*DB2*(fub+fsb)+S1*S2*(fub+fdb))

        SIG2=WQ4*(U1*UB2*fu+D1*DB2*fd+S1*S2*fs+UB1*U2*fub+DB1*D2*fdb+S1*S2*fsb) \
            +WT4*(UB1*U2*fu+DB1*D2*fd+S1*S2*fs+U1*UB2*fub+D1*DB2*fdb+S1*S2*fsb) \
            +WQ1*(U1*(D2+DB2+S2+S2)*fu+D1*(U2+UB2+S2+S2)*fd+UB1*(D2+DB2+S2+S2)*fub \
            +DB1*(U2+UB2+S2+S2)*fdb+S1*(U2+UB2+D2+DB2)*fs+S1*(U2+UB2+D2+DB2)*fsb)

        SIG3=WT1*((D1+DB1+S1+S1)*U2*fu+(U1+UB1+S1+S1)*D2*fd \
            +(D1+DB1+S1+S1)*UB2*fub+(U1+UB1+S1+S1)*DB2*fdb \
            +(U1+UB1+D1+DB1)*S2*fs+(U1+UB1+D1+DB1)*S2*fsb) \
            +WQ5*(U1*UB2+D1*DB2+S1*S2)*fg+WT5*(UB1*U2+DB1*D2+S1*S2)*fg

        SIG4=WQ6*GL1*GL2*(fu+fd+fs)+WT6*GL1*GL2*(fub+fdb+fsb) \
            +WQ7*(GL2*(U1*fu+D1*fd+S1*fs+UB1*fub+DB1*fdb+S1*fsb) \
            +GL1*fg*(U2+UB2+D2+DB2+S2+S2)) \
            +WT7*(GL1*(U2*fu+D2*fd+S2*fs+UB2*fub+DB2*fdb+S2*fsb) \
            +GL2*fg*(U1+UB1+D1+DB1+S1+S1)) \
            +WQ8*GL1*GL2*fg

        SIG=SIG1+SIG2+SIG3+SIG4

        sigma = 1./xa/(zc*zc)/BB*SIG*alpha(scale)**2/roots2
    else:
        sigma = 0.

    return sigma
```

Let us now set up the vegas integration

1. We will generate the differential cross section as a function of transverse momentum p_T, so we integrate over x_a, z_c, y.
2. We will compare with data from PHENIX collaboration at RHIC, which integrate over $-0.35 < y < 0.35$. They defined as a normalized rapidity distribution, this is why we divide by the rapidity interval $dY = y_{\max} - y_{\min} = 0.7$.

```
import vegas
```

```
def dist_vegas(roots, pT, scale):
    convert = 0.389
    def cross_before(val):
        xa = val[0]
        zc = val[1]
        y  = val[2]
        return sigma(xa, zc, y, roots, pT, scale)

    ymin = -0.35; ymax = 0.35; dY = ymax - ymin
    integ = vegas.Integrator([[0, 1], [0, 1], [ymin, ymax]])
    result = integ(cross_before, nitn=10, neval=10000)
    return convert/dY*result.mean
```

Computing

```
roots = 200.
pT_a = [1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5, 10.5, 11.5, 12.5, 13.5, 14.5, 15.5, 16.5, 17.5, 18.5, 19.5]
num0 = len(pT_a)
cross = [0.]*num0
for i in range(num0):
    pT = pT_a[i]
    scale = pT
    cross[i] = dist_vegas(roots, pT, scale)
    print('%0.3e\t %0.3e' %(pT, cross[i]))
```

```
1.500e+00        7.848e-02
2.500e+00        3.008e-03
3.500e+00        2.909e-04
4.500e+00        4.708e-05
5.500e+00        1.055e-05
```

```
6.500e+00        2.924e-06
7.500e+00        9.551e-07
8.500e+00        3.490e-07
9.500e+00        1.411e-07
1.050e+01        6.122e-08
1.150e+01        2.827e-08
1.250e+01        1.378e-08
1.350e+01        7.005e-09
1.450e+01        3.688e-09
1.550e+01        2.013e-09
1.650e+01        1.130e-09
1.750e+01        6.478e-10
1.850e+01        3.807e-10
1.950e+01        2.280e-10
```

Get ready for plots

1. Export the theory calculations to a file using pandas
2. Set plot features, e.g. font, font size, etc.
3. Make sure to change Google Colab directory to your Google Drive folder to store and call files

```
# from google.colab import drive
# drive.mount('/content/drive')
# import sys
# # sys.path.append('/content/drive/MyDrive/ColabNotebooks/EICschool')
# dir= '/content/drive/MyDrive/ColabNotebooks/EICschool'
# !ls $dir
```

```
# !cat $dir/python_thy.csv
```

```
# dir
```

```
# # experimental data, to compare with theory
# data = pd.read_csv(dir+'/python_thy.csv',sep=",", comment='#')
# # data = pd.read_csv('/content/drive/MyDrive/ColabNotebooks/LHAPDF/python_thy.csv')
# df=pd.DataFrame(data)
# print(df.shape)
# print(df.head())
# df['pT']
# df['cross']
```

```
import io
# Your raw text
textdata="""pT,cross
1.5,0.07867554878006042
2.5,0.002990794193633156
3.5,0.0002910374889493148
4.5,4.709642402131125e-05
5.5,1.0550984199278837e-05
6.5,2.928455745171195e-06
7.5,9.546423919328887e-07
8.5,3.5080548595319374e-07
9.5,1.4064481255342338e-07
10.5,6.133965406255002e-08
11.5,2.8327466228132276e-08
12.5,1.3772283092492418e-08
13.5,6.986563558983835e-09
14.5,3.6935527243574332e-09
15.5,2.0154678391532878e-09
16.5,1.1264156983698888e-09
17.5,6.478672037565893e-10
18.5,3.799513324580145e-10
19.5,2.277201933803674e-10"""


# Use StringIO to parse the string like a file
df2 = pd.read_csv(io.StringIO(textdata))
print(df2)
```

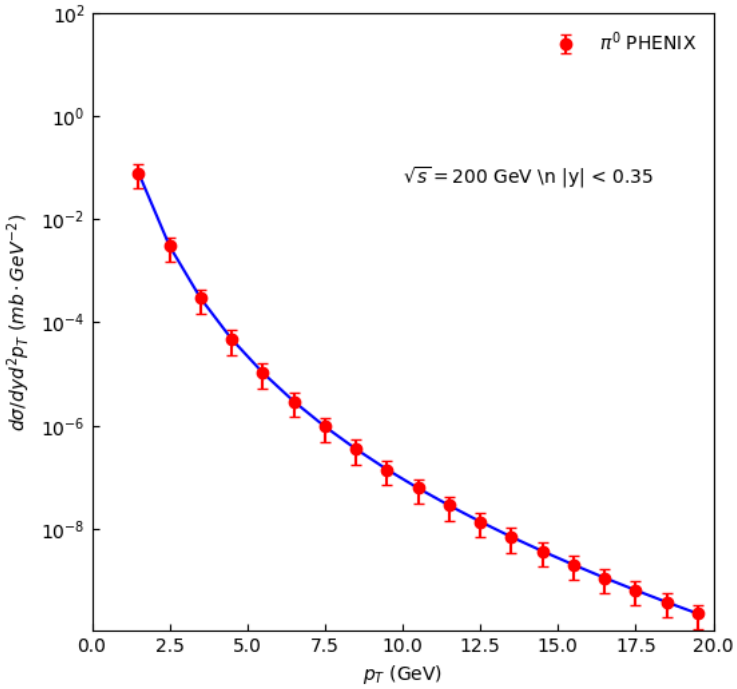
```
      pT         cross
0    1.5  7.867555e-02
1    2.5  2.990794e-03
2    3.5  2.910375e-04
3    4.5  4.709642e-05
4    5.5  1.055098e-05
5    6.5  2.928456e-06
6    7.5  9.546424e-07
7    8.5  3.508055e-07
8    9.5  1.406448e-07
```

```
9   10.5  6.133965e-08
10  11.5  2.832747e-08
11  12.5  1.377228e-08
12  13.5  6.986564e-09
13  14.5  3.693553e-09
14  15.5  2.015468e-09
15  16.5  1.126416e-09
16  17.5  6.478672e-10
17  18.5  3.799513e-10
18  19.5  2.277202e-10
```

```
df=df2
```

```
# experimental data, to compare with theory
# data = pd.read_csv(dir+'/python_thy.csv',sep=",", comment='#')
# # data = pd.read_csv('/content/drive/MyDrive/ColabNotebooks/LHAPDF/python_thy.csv')
# df=pd.DataFrame(data)
print(df.shape)
print(df.head())

x = df['pT']
y = df['cross']
print(x.head())
print(y.head())



num2 = len(x)
# yerr = np.sqrt(data[2]**2+data[3]**2)
# FAKE ERRORS:
error=0.50
yerr=error*y


for i in range(num2):
    print('%0.3e\t %0.3e\t %0.3e' %(x[i], y[i], yerr[i]))

plt.figure(figsize=(6, 6))
plt.yscale('log')
plt.tick_params(axis='both', which='both', direction='in')
plt.xlabel('$p_T$ (GeV)')
plt.ylabel('$d\\sigma/dyd^2p_T$ ($mb\\cdot GeV^{-2}$)')
plt.xlim(0,20)
plt.ylim(1.1e-10,1e2)

# plot experimental data points
plt.errorbar(x, y, yerr = yerr, fmt = 'ro', label='$\\pi^0$ PHENIX', capsize=3, markersize = 6)

# plot theory as curve
# x_val = python_thy["pT"]
x_val = x
# over here, I added this "multiply()" function to just show
# that pandas has this feature to multiply each element of the column
# y_val = python_thy["cross"].multiply(1.0)
y_val =  y
plt.plot(x_val, y_val, 'b-')

plt.legend(frameon=False)
plt.text(10, 0.05, '$\\sqrt{s} = 200$ GeV \\n |y| < 0.35')
plt.savefig('RHIC_pp.pdf', bbox_inches='tight')
plt.show()
```

```
(19, 2)
    pT     cross
0  1.5  0.078676
1  2.5  0.002991
2  3.5  0.000291
3  4.5  0.000047
4  5.5  0.000011
0    1.5
1    2.5
2    3.5
3    4.5
4    5.5
Name: pT, dtype: float64
0    0.078676
1    0.002991
2    0.000291
3    0.000047
4    0.000011
Name: cross, dtype: float64
1.500e+00   7.868e-02  3.934e-02
2.500e+00   2.991e-03  1.495e-03
3.500e+00   2.910e-04  1.455e-04
4.500e+00   4.710e-05  2.355e-05
5.500e+00   1.055e-05  5.275e-06
6.500e+00   2.928e-06  1.464e-06
7.500e+00   9.546e-07  4.773e-07
8.500e+00   3.508e-07  1.754e-07
9.500e+00   1.406e-07  7.032e-08
1.050e+01   6.134e-08  3.067e-08
1.150e+01   2.833e-08  1.416e-08
1.250e+01   1.377e-08  6.886e-09
1.350e+01   6.987e-09  3.493e-09
1.450e+01   3.694e-09  1.847e-09
1.550e+01   2.015e-09  1.008e-09
1.650e+01   1.126e-09  5.632e-10
1.750e+01   6.479e-10  3.239e-10
1.850e+01   3.800e-10  1.900e-10
1.950e+01   2.277e-10  1.139e-10
```




```
 1 import io
 2 # Your raw text
 3 textdata="""pT,cross
 4 1.5,0.07867554878006042
 5 2.5,0.002990794193633156
 6 3.5,0.0002910374889493148
 7 4.5,4.709642402131125e-05
 8 5.5,1.0550984199278837e-05
 9 6.5,2.928455745171195e-06
10 7.5,9.546423919328887e-07
11 8.5,3.5080548595319374e-07
12 9.5,1.4064481255342338e-07
13 10.5,6.133965406255002e-08
14 11.5,2.8327466228132276e-08
15 12.5,1.3772283092492418e-08
16 13.5,6.986563558983835e-09
17 14.5,3.6935527243574332e-09
18 15.5,2.0154678391532878e-09
```

```
16.5,1.1264156983698888e-09
17.5,6.478672037565893e-10
18.5,3.799513324580145e-10
19.5,2.277201933803674e-10"""


# Use StringIO to parse the string like a file
df2 = pd.read_csv(io.StringIO(textdata))
print(df2)
```

```
      pT         cross
0    1.5  7.867555e-02
1    2.5  2.990794e-03
2    3.5  2.910375e-04
3    4.5  4.709642e-05
4    5.5  1.055098e-05
5    6.5  2.928456e-06
6    7.5  9.546424e-07
7    8.5  3.508055e-07
8    9.5  1.406448e-07
9   10.5  6.133965e-08
10  11.5  2.832747e-08
11  12.5  1.377228e-08
12  13.5  6.986564e-09
13  14.5  3.693553e-09
14  15.5  2.015468e-09
15  16.5  1.126416e-09
16  17.5  6.478672e-10
17  18.5  3.799513e-10
18  19.5  2.277202e-10
```

```
# experimental data, to compare with theory
# data = pd.read_csv(dir+'/python_thy.csv',sep=",", comment='#')
# data = pd.read_csv('/content/drive/MyDrive/ColabNotebooks/LHAPDF/python_thy.csv')
# df=pd.DataFrame(data)
df=df2
print(df.shape)
print(df.head())

x = df['pT']
y = df['cross']
print(x.head())
print(y.head())


num2 = len(x)
# yerr = np.sqrt(data[2]**2+data[3]**2)
# FAKE ERRORS:
error=0.50
yerr=error*y


for i in range(num2):
    print('%0.3e\t %0.3e\t %0.3e' %(x[i], y[i], yerr[i]))

plt.figure(figsize=(6, 6))
plt.yscale('log')
plt.tick_params(axis='both', which='both', direction='in')
plt.xlabel('$p_T$ (GeV)')
plt.ylabel('$d\\sigma/dyd^2p_T$ ($mb\\cdot GeV^{-2}$)')
plt.xlim(0,20)
plt.ylim(1.1e-10,1e2)

# plot experimental data points
plt.errorbar(x, y, yerr = yerr, fmt = 'ro', label='$\\pi^0$ PHENIX', capsize=3, markersize = 6)

# plot theory as curve
# x_val = python_thy["pT"]
x_val = x
# over here, I added this "multiply()" function to just show
# that pandas has this feature to multiply each element of the column
# y_val = python_thy["cross"].multiply(1.0)
y_val =  y
plt.plot(x_val, y_val, 'b-')

plt.legend(frameon=False)
plt.text(10, 0.05, '$\\sqrt{s} = 200$ GeV \\n |y| < 0.35')
plt.savefig('RHIC_pp.pdf', bbox_inches='tight')
plt.show()
```

```
(19, 2)
    pT     cross
0  1.5  0.078676
1  2.5  0.002991
2  3.5  0.000291
3  4.5  0.000047
4  5.5  0.000011
0    1.5
1    2.5
2    3.5
3    4.5
4    5.5
Name: pT, dtype: float64
0    0.078676
1    0.002991
2    0.000291
3    0.000047
4    0.000011
Name: cross, dtype: float64
1.500e+00   7.868e-02  3.934e-02
2.500e+00   2.991e-03  1.495e-03
3.500e+00   2.910e-04  1.455e-04
4.500e+00   4.710e-05  2.355e-05
5.500e+00   1.055e-05  5.275e-06
6.500e+00   2.928e-06  1.464e-06
7.500e+00   9.546e-07  4.773e-07
8.500e+00   3.508e-07  1.754e-07
9.500e+00   1.406e-07  7.032e-08
1.050e+01   6.134e-08  3.067e-08
1.150e+01   2.833e-08  1.416e-08
1.250e+01   1.377e-08  6.886e-09
1.350e+01   6.987e-09  3.493e-09
1.450e+01   3.694e-09  1.847e-09
1.550e+01   2.015e-09  1.008e-09
1.650e+01   1.126e-09  5.632e-10
1.750e+01   6.479e-10  3.239e-10
1.850e+01   3.800e-10  1.900e-10
1.950e+01   2.277e-10  1.139e-10
```